Numeric Factorization of Symmetric Positive Definite Matrices

Vanessa López
CS497 MH
Spring 1999

Outline

- Cholesky Factorization
- Observations about the Cholesky Factorization Algorithm
- Forms of Cholesky Factorization
- Data Dependence in Column-Oriented Forms
- Cost of Computing the Cholesky Factor
- Sparse Column Cholesky Algorithms

Cholesky Factorization

- Every n-by-n symmetric positive definite (SPD) matrix A can be factored as
$$A = LL^T,$$
where L is a lower triangular matrix with positive diagonal elements.

- L is called the *Cholesky factor* of A.

- Solution to the system $Ax = b$ is obtained by solving
$$\begin{aligned} Ly &= b \qquad \text{(forward substitution)} \\ L^T x &= y \qquad \text{(back substitution)} \end{aligned}$$

- Cholesky factorization algorithm:
  - Variant of Gaussian elimination.
  - Takes advantage of symmetry to reduce work and storage by about half.
  - Consists of a triple nested loop, so there are 3! ways of arranging it.

Some Observations About the Algorithm

- Direct computation of $A = LL^T$ gives, for each column $j = 1, 2, \ldots, n$,

$$
\begin{aligned}
l_{jj} &= (a_{jj} - \Sigma_{k=1}^{j-1} l_{jk}^2)^{1/2} && (1) \\
l_{ij} &= (a_{ij} - \Sigma_{k=1}^{j-1} l_{ik} l_{jk})/l_{jj}, \quad i = j+1, \ldots, n && (2)
\end{aligned}
$$

- Can be written as:

$$
\begin{array}{l}
\textbf{for} \;\; j = 1,\ n \\
\quad \textbf{for} \;\; k = 1,\ j-1 \\
\quad\quad \textbf{for} \;\; i = j,\ n \\
\quad\quad\quad a_{ij} = a_{ij} - a_{ik} \cdot a_{jk} \qquad \{cmod(j,k)\} \\
\quad a_{jj} = \sqrt{a_{jj}} \\
\quad \textbf{for} \;\; k = j+1,\ n \\
\quad\quad a_{kj} = a_{kj}/a_{jj} \qquad\qquad \{cdiv(j)\}
\end{array}
$$

Observations:

- Pivoting is not required for numerical stability.

  – From (1) we have

$$a_{jj} = \Sigma_{k=1}^{j-1} l_{jk}^2 + l_{jj}^2 = \Sigma_{k=1}^{j} l_{jk}^2,$$

  so the elements of L are bounded by $\max\{\sqrt{a_{jj}}\}$.

Observations, continued

- The square roots in (1) are of positive numbers, so the algorithm is well defined.
  - For $A = L_G U_G$ (LU decomposition of A) we have $\det A_k = \prod u_{ii}$, where A_k is the k-by-k matrix consisting of the first k rows and k columns of A.
  - This implies that Gaussian elimination can be carried out without row or column interchanges iff $\det A_k \neq 0$, $k = 1, \ldots, n-1$.
  - A symmetric matrix B is positive definite iff $\det B_k > 0$, $k = 1, \ldots, n$.
  - So the pivots in (1) are positive iff A is SPD.

- Only the lower triangular part of A is accessed.

- L is computed in place, overwriting the lower triangle of A.

- If the multiple a_{jk} in the $cmod(j, k)$ operation is zero, the innermost loop has no effect.

- Fill is introduced.

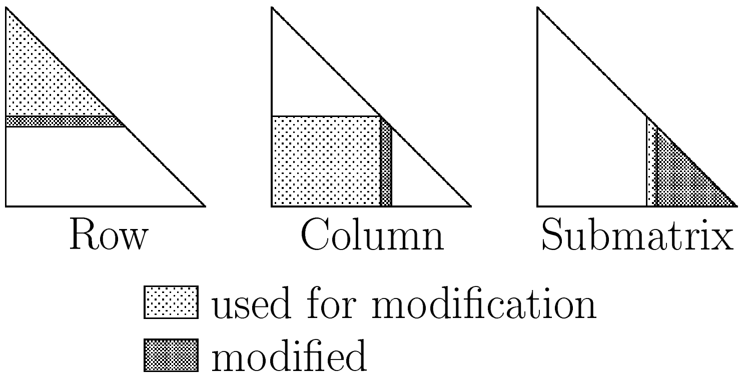
Three Forms of Cholesky Factorization

- Choice of outer loop index yields different memory access patterns.

- Impact performance in architectural features such as cache utilization, vectorization, and parallelism.

- Forms:
  - *Row-Cholesky:* Row counter is in the outer loop. The inner loops solve a triangular system for each new row in terms of the previously computed rows.
  - *Column-Cholesky:* Successive columns of L are computed one by one. The inner loops compute a matrix-vector product giving the effect that the previously computed columns have on the column currently being computed.
  - *Submatrix-Cholesky:* Successive columns of L are computed one by one. The inner loops apply the current column as a rank-1 update to the remaining unreduced submatrix.

Three Forms of Cholesky Factorization, continued



Column-Cholesky:

- left-looking
- delayed-update
- demand-driven
- fan-in

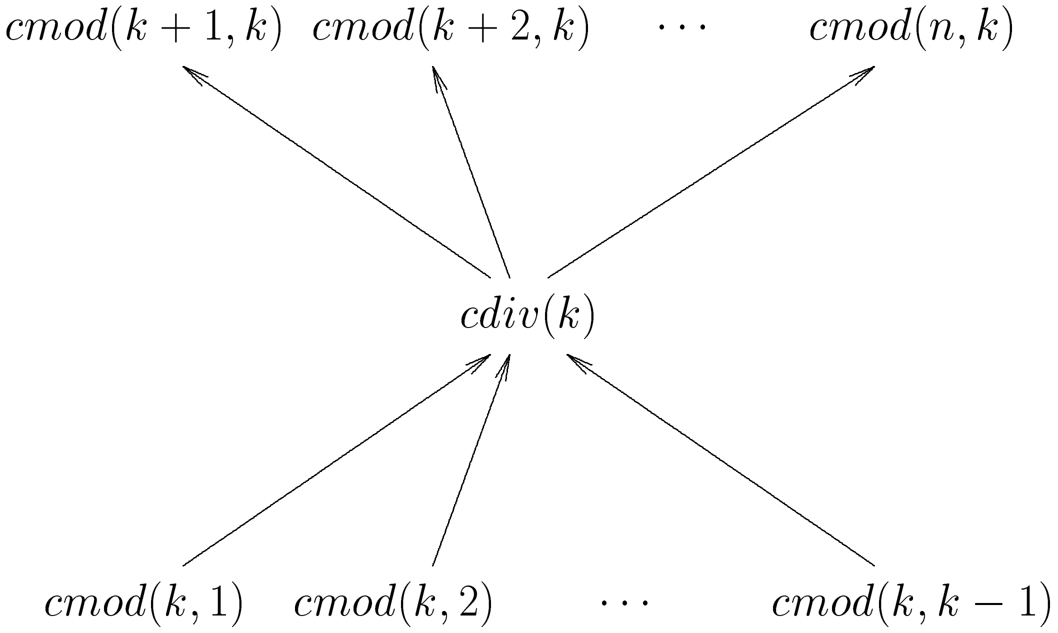
Submatrix-Cholesky:

- right-looking
- immediate-update
- data-driven
- fan-out

Notation

- $cmod(j,k)$: column j is modified by a multiple of column k.
- $cdiv(j)$: scaling of column j by the square root of its diagonal element.
- $Struct(M_{i*})$: $\{k < i | m_{ik} \neq 0\}$ (i.e., sparsity structure of row i of the strict lower triangle of the matrix M.
- $Struct(M_{*j})$: $\{k > j | m_{kj} \neq 0\}$ (i.e., sparsity structure of column j of the strict lower triagle of the matrix M.

Data Dependence in Column-Oriented Forms



Observations:

- $cmod(k,*)$ operations along the bottom can be done in any order, but they all have the same target column. The updating must be coordinated to preserve data integrity.
- $cmod(*,k)$ operations along the top can be done in any order. They all have different target columns, so the updating can be done simultaneously.

Data Dependence in Column-Oriented Forms, continued

- Performing *cmods* concurrently is the most important source of parallelism in column-oriented factorization algorithms.
- For a dense matrix, each $cdiv(k)$ depends on the immediately preceding column, so only one *cdiv* can be done at a time.
- Sparse matrices potentially have an additional source of parallelism, since "missing" *cmods* may permit multiple *cdiv*s to be done simultaneously.

Cost of Computing Cholesky Factor L

- $\eta(\diamond)$: number of nonzero components in $\diamond$.
- (George & Liu)
  The number of multiply/divide operations required to compute the triangular factor L of the n-by-n matrix A is
  $$\frac{1}{2}\sum_{i=1}^{n-1}(\eta(L_{*i})-1)(\eta(L_{*i})+2)$$
  (similar number of add/subtract.)
- The number of operations is determined by the sparsity structure of the strict lower part of L.

Sparse Column Cholesky Algorithms

Sparse column-Cholesky factorization

$$
\begin{array}{l}
\textbf{for}\ \ j = 1,\ n \\
\quad \textbf{for}\ \ k \in Struct(L_{j*}) \\
\quad\quad cmod(j,k) \\
\quad cdiv(j)
\end{array}
$$

Sparse submatrix-Cholesky factorization

$$
\begin{array}{l}
\textbf{for}\ \ k = 1,\ n \\
\quad cdiv(k) \\
\quad \textbf{for}\ \ j \in Struct(L_{*k}) \\
\quad\quad cmod(j,k)
\end{array}
$$

Cholesky Factorization and Elimination Trees

- The elimination tree displays the data dependences among the columns of the Cholesky factor.

- Each column of L depends only on its descendants in the elimination tree.

- Leaves in the tree represent processes that may execute in parallel (there is no contention for write access to a common location).

- The height of the elimination tree determines the longest serial path through the computation.

- The width of the tree determines the degree of parallelism available (number of processors that can be used simultaneoulsy).

- A short, wide, well-balanced elimination tree is desirable for parallel computation.

Levels of Parallelism Available in Cholesky Factorization

- *Fine-grain*: each task consists of only one or two floating point operations, such as a multiply-add pair.

- *Medium-grain*: each task is a single column operation, such as *cmod* or *cdiv*.

- *Large-grain*: each task is the computation of an entire group of columns in a subtree of the elimination tree.

- Fine-grain parallelism is available in either the dense or sparse case. Not easily exploited by most current generation parallel computers because of high communication overhead.

- Medium-grain parallelism is also available in either the dense or sparse case. Performing many *cmod* operations simultaneously accounts for essentially all of the parallelism typically exploited in dense factorization on most parallel computers.

- Large-grain parallelism is available only in sparse case. Only in the sparse case can multiple *cdiv* operations potentially be done simultaneously.

References

G. Dahlquist and A. Bjorck, Numerical Methods, Prentice-Hall, 1974.

J. W. Demmel, M. T. Heath, and H. A. van der Vorst, Parallel Numerical Linear Algebra, *Acta Numerica*, 2 (1993), pp. 111–197.

J.. Dongarra, I. Duff, D. Sorensen, and H. A. van der Vorst, Solving Linear Systems on Vector and Shared Memeory Computers, SIAM Publications, 1991.

J. A. George and J. W-H. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, 1981.